Activty recognition from single chest mounted accelerometer

David Janda

11 11 2020

Abstaract

Introduction

Main goal of this data science project is to build at least two machine learning algorithms for Activity recognition from single chest mounted accelerometer data. Target accuracy of best performing algorithm is 95 % or more. The data set is acquired through UCI machine learning repository. The data consist of 1 file per user with 15 users total. Each file contains data for triaxial accelerometer, sequential number and user id.

Methods

The data from each user are combined together to make one dataset. The set is then cleaned and separated into data and validation (unseen) set. Data exploration is done through written or visualized form. For visualization are used various packages.
There are three algorithms build - decision tree, random forest and k-nearest neighbors. After is the algorithm build it's then fit into the validation set.

Results

The best results were acquired by Random forest algorithm with accuracy of 98 %. For this approach was used Ranger package with auto tuning parameters.

Conclusion

Our task was to build machine learning algorithm for human activity recognition. It was done by three different approaches. And best accuracy was acquired by random forest algorithm. The results could be improved by setting different algorithms together like kNN + Rf. Or by using neural network or deep learning techniques.
Limitations of this work could be limited knowledge of accelerometers used for data measurement, therefore the algorithm wouldn't have to work on different data set.

Introduction

Human activity recognition also called HAR is getting a lot of attention since it's inception. And this field is still evolving. The knowledge of user's activity can help to personalize services, improve health care and study human behavior such as in sociology or physical activity science. Since nowadays there were many sensor based devices used to monitor human activity. Such as heart rate monitors, GPS or accelerometers. From the heart rate monitors and GPS signals we can only roughly determine what kind of activity is person doing and we have to contextualize it with environment in which is the activity done. Or use other devices that can help us to understand the persons movement. One of these devices are accelerometers, which is an electromechanical device used to measure acceleration forces. The force may be static, like gravitational or dynamic, like human movement. The accelerometers that are used for monitoring humans movement are usually small devices, that can be attached for example to wrist, waist or chest. However there are also accelerometers in smart phones and their signal can also be used for activity recognition. The device can measure the activity in different axis, nowadays is the standard to measure in triaxial dimension - x, y, z. The signal is then stored in device and can be downloaded a transformed into csv. file format. However for human activity recognition we need some base data that are labeled with specific activity so we can use them to train algorithms. Therefore for our purpose we will use data from UC Irvine Machine Learning Repository.

Dataset

We will use data provided for **Activity recognition from single chest mounted accelerometer**, the data are obtained from this website: http://archive.ics.uci.edu/ml/datasets/Activity+Recognition+from+Single+Chest-Mounted+Accelerometer# Downloaded folder consist data from 15 participants, one .csv file per person, with label of activity type. Activities performed are: 1. Working at Computer
2. Standing Up, Walking and Going updown stairs
3. Standing
4. Walking
5. Going UpDown Stairs
6. Walking and Talking with Someone
7. Talking while Standing

The labels are codified as numbers. The data of three axes are recorded on sampling frequency of 52 Hz and are uncalibrated. The data set also contains sequential number.

Goals of this project

This project is last step in HarvardX PH125.9x Data Science professional certificate and part of Capstone course. Aim of this project is to **build at least two machine learning algorithms** that will predict human activity based on input from triaxial accelerometer. The results will be tested on generated validation set. Performance of the algorithm will be evaluated by accuracy. Personal goal is to have accuracy better than 95 %.

Steps toward the goal

- Introduction
- Data downloading and cleaning
- Description of dataset
- Exploration analysis
- Building algorithm for activity prediction
- Activity prediction
- Summarize and give recommendations for future work

Methods

Preparation of work space

First we load necessary packages that are going to be used, if they are not installed we will install them:

```
# install packages if they are not installed
if (!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if (!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if (!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if (!require(knitr)) install.packages("knitr", repos = "http://cran.us.r-project.org")
if (!require(ggridges)) install.packages("ggridges", repos = "http://cran.us.r-project.org")
if (!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if (!require(e1071)) install.packages("e1071", dependencies = TRUE, repos = "http://cran.us.r-project.o
if (!require(stats)) install.packages("stats", repos = "http://cran.us.r-project.org")
if (!require(gridExtra)) install.packages("gridExtra", repos = "http://cran.us.r-project.org")
if (!require(ranger)) install.packages("ranger", repos = "http://cran.us.r-project.org")
if (!require(tuneRanger)) install.packages("tuneRanger", repos = "http://cran.us.r-project.org")
if (!require(forecast)) install.packages("forecast", dependencies = TRUE, repos = "http://cran.us.r-pro
if (!require(doParallel)) install.packages("doParallel", repos = "http://cran.us.r-project.org")

# Load them if not loaded
if (!require(tidyverse)) library("tidyvers")
if (!require(caret)) library("caret")
if (!require(data.table)) library("data.table")
if (!require(knitr)) library("knitr")
if (!require(ggridges)) library("ggridges")
if (!require(ggplot2)) library("ggplot2")
if (!require(e1071)) library("e1071")
if (!require(stats)) library("stats")
if (!require(gridExtra)) library("gridExtra")
if (!require(ranger)) library("stats")
if (!require(tuneRanger)) library("stats")
if (!require(forecast)) library("forecast")
if (!require(doParallel)) library("forecast")
```

Then we can download our data from https://archive.ics.uci.edu/ml/datasets/Activity+Recognition+from+Single+Chest-Mounted+Accelerometer#. We download the zipped file into temporary object.

As the data were downloaded in separated files per user we will merge them into one dataset.Also it has been said that the data are uncalibrated so we will clear them from noise with median filter and normalize theme so the values are on the same scale. We will also change the sequential number so the activities can be displayed next to each other. Then we create new value named vector magnitude which represents length of vector for our three axis.

In the dataset was found activity category 0 which is not described, so we delete it. Also we give meaningful labels to activities.

```
# Crating one data frame
AR <- bind_rows(list(AR1, AR2, AR3, AR4, AR5, AR6, AR7, AR8, AR9, AR10, AR11, AR12, AR13, AR14, AR15),

# Adjustment of data
## Giving meaningful label to activities
## Also delete activity = 0, because there wasn't specification about this type
```

```r
AR$activity <- factor(AR$activity,
                      levels = c(1,2,3,4,5,6,7),
                      labels = c("working", "stand_walk_down",
                                 "standing", "walking",
                                 "going_up_down", "walk+talk", "stand+talk"))


# Creating normalization function

normalize <- function(x){
  return(x - min(x)) / (max(x) - min(x))
}


## Creating vector magnitude, applying median filter to remove outlines, normalizing data
## Creating sequential number for each activity

AR <- AR %>%
  mutate(id = as.numeric(id),
         vm = sqrt(x^2 + y^2 + z^2),
          x = runmed(x, k = 5),
          y = runmed(y, k = 5),
          z = runmed(z, k = 5)) %>%
         group_by(activity) %>%
         mutate(seq = as.numeric(1:n()),
                x = normalize(x),
                y = normalize(y),
                z = normalize(z),
                vm = normalize(vm)) %>%
  subset(activity != 0)
```

As last step of our preparation we split the data into part that we will use for visualization and algorithm building and part for validation of our algorithm.
The splitting is done 70/30 to have major part of the data for manipulation.

Data analysis

Dataset

To know with what kind of data are we dealing with we will now display some describing parameters.
Let's start with dimensions. Number of rows and column.

```
## [1] 1346222       7
```

As we can see the data has 7 columns, each for one variable and a lot of rows.
It's understandable because it's signal with frequency of 52 Hz. For more details we display first 6 rows.

```
## # A tibble: 6 x 7
##      id   seq     x     y     z activity    vm
##   <dbl> <dbl> <dbl> <dbl> <dbl> <fct>    <dbl>
## 1     1     2   968  1687  2038 working  1945.
## 2     1     3   968  1580  1897 working  1761.
## 3     1     4   968  1574  1897 working  1700.
## 4     1     5   968  1572  1897 working  1766.
## 5     1     8   961  1574  1912 working  1732.
## 6     1     9   964  1593  1931 working  1786.
```

The signal is not much understandable if we don't know it's composition.
Lets see what kind of values do we have on each axes and other variables with summary() and str() function:

```
# Summary of data
data %>% summary()
```

```
##        id              seq               x                y         
##  Min.   : 1.000   Min.   :     1   Min.   :   0.0   Min.   :   0.0  
##  1st Qu.: 4.000   1st Qu.: 83474   1st Qu.: 388.0   1st Qu.: 463.0  
##  Median : 7.000   Median :203603   Median : 534.0   Median : 633.0  
##  Mean   : 7.513   Mean   :235203   Mean   : 724.2   Mean   : 990.8  
##  3rd Qu.:11.000   3rd Qu.:360927   3rd Qu.:1248.0   3rd Qu.:1949.0  
##  Max.   :15.000   Max.   :608667   Max.   :2886.0   Max.   :3709.0  
##                                                                     
##        z                     activity            vm        
##  Min.   :   0.0   working        :426154   Min.   :   0.0  
##  1st Qu.: 360.0   stand_walk_down: 33550   1st Qu.: 668.5  
##  Median : 442.0   standing       :152078   Median : 790.6  
##  Mean   : 882.5   walking        :249838   Mean   :1189.1  
##  3rd Qu.:1846.0   going_up_down  : 36035   3rd Qu.:2157.0  
##  Max.   :4084.0   walk+talk      : 33305   Max.   :4625.1  
##                   stand+talk     :415262                   
```

```
# Structure of data
data %>% str()
```

```
## tibble [1,346,222 x 7] (S3: grouped_df/tbl_df/tbl/data.frame)
##  $ id      : num [1:1346222] 1 1 1 1 1 1 1 1 1 1 ...
##  $ seq     : num [1:1346222] 2 3 4 5 8 9 10 11 12 13 ...
##  $ x       : num [1:1346222] 968 968 968 968 961 964 953 947 953 953 ...
```

```
##  $ y       : num [1:1346222] 1687 1580 1574 1572 1574 ...
##  $ z       : num [1:1346222] 2038 1897 1897 1897 1912 ...
##  $ activity: Factor w/ 7 levels "working","stand_walk_down",..: 1 1 1 1 1 1 1 1 1 ...
##  $ vm      : num [1:1346222] 1945 1761 1700 1766 1732 ...
##  - attr(*, "groups")= tibble [7 x 2] (S3: tbl_df/tbl/data.frame)
##   ..$ activity: Factor w/ 7 levels "working","stand_walk_down",..: 1 2 3 4 5 6 7
##   ..$ .rows   : list<int> [1:7]
##   .. ..$ : int [1:426154] 1 2 3 4 5 6 7 8 9 10 ...
##   .. ..$ : int [1:33550] 23496 23497 23498 23499 23500 23501 23502 23503 23504 23505 ...
##   .. ..$ : int [1:152078] 24152 24153 24154 24155 24156 24157 24158 24159 24160 24161 ...
##   .. ..$ : int [1:249838] 26877 26878 26879 26880 26881 26882 26883 26884 26885 26886 ...
##   .. ..$ : int [1:36035] 48614 48615 48616 48617 48618 48619 48620 48621 48622 48623 ...
##   .. ..$ : int [1:33305] 52950 52951 52952 52953 52954 52955 52956 52957 52958 52959 ...
##   .. ..$ : int [1:415262] 54987 54988 54989 54990 54991 54992 54993 54994 54995 54996 ...
##   .. ..@ ptype: int(0)
##   ..- attr(*, ".drop")= logi TRUE
```

For our analysis we need clear dataset without missing values so we check also this:

```
# Missing values check
sum(is.na(data))
```

```
## [1] 0
```

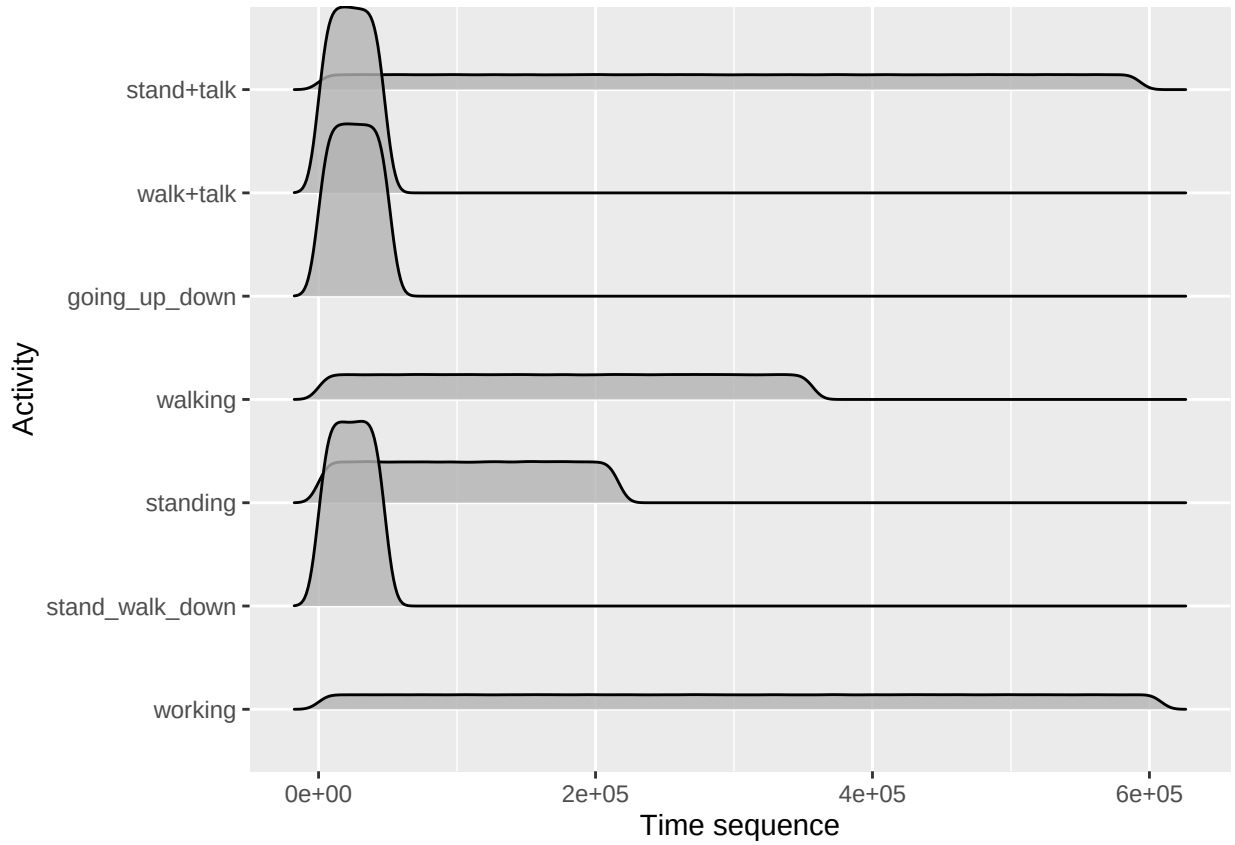
```
sum(is.na(validation))
```

```
## [1] 0
```

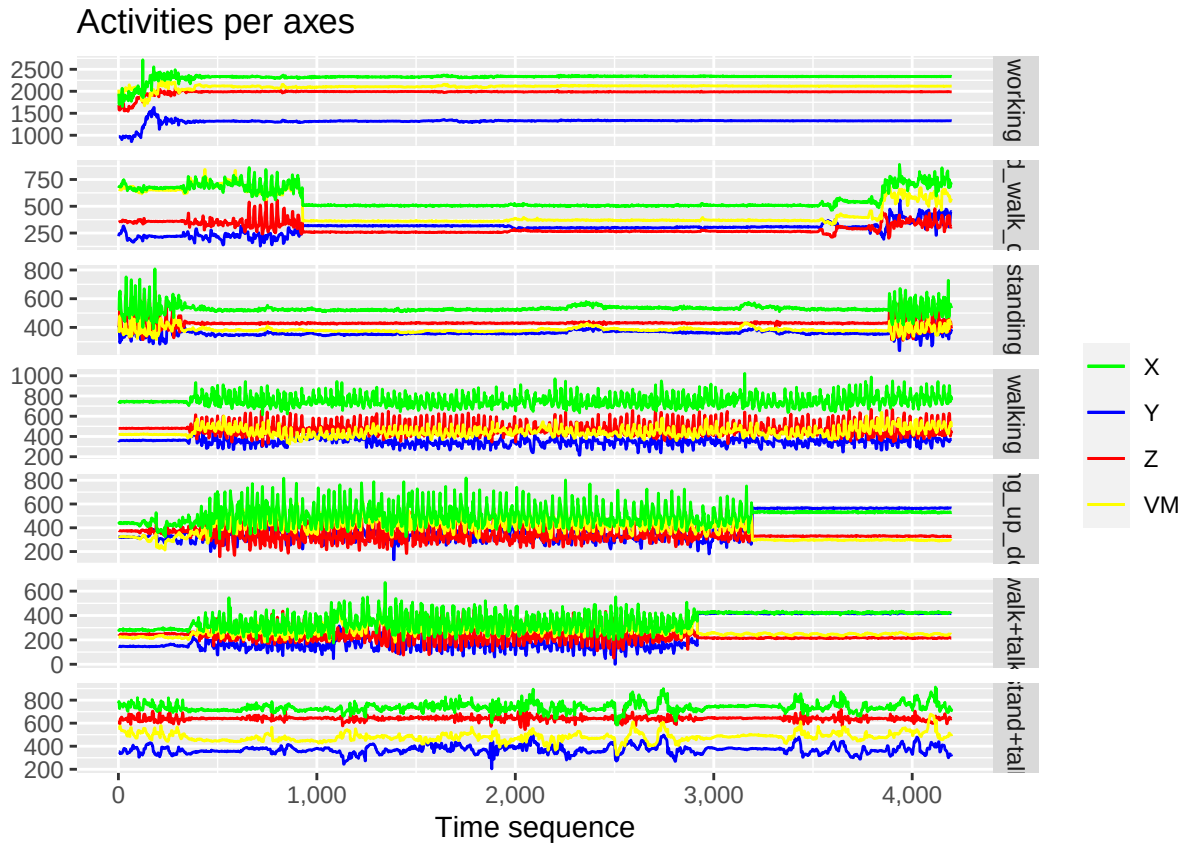
Now we know with what kind of data are we working. So we can go on and explore each activity signal.

Activities

We know there are 7 activities with labels to differ them. We will start with plotting their length:



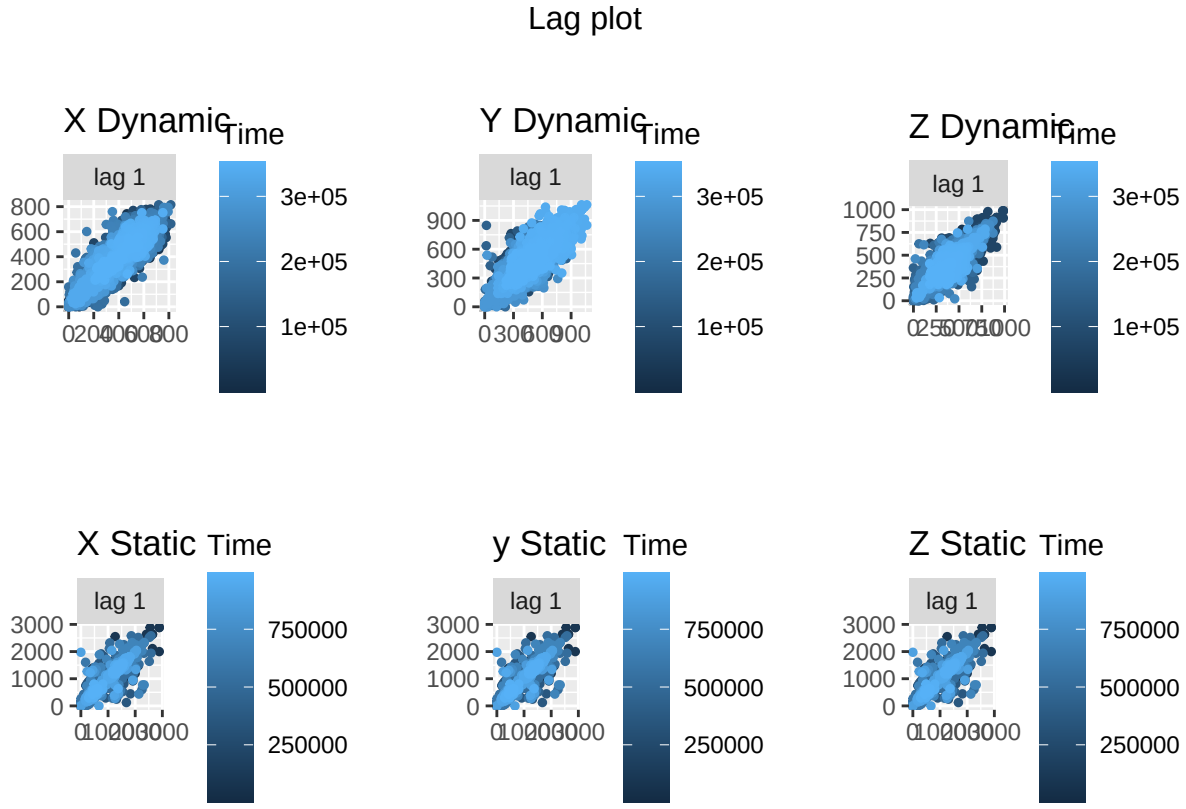
As we can see the activities have different duration, however it shouldn't affect our analysis.
We will now plot signal per activity in smaller time window so we can see if there is any pattern to recognize.



As we can see the difference in patterns is quiet visible. For example in walking we can see the waving of curve because of the steps, the same goes for going up or down, walking + talking and stand + walk down. On the other hand the static activities as working or standing seems to have more straight line signal. Also we can see there is difference in strength of the signal.
From this graph we can see that it shouldn't be hard to recognize the activity. For example the signal of working is much more stable but stronger, the standing is also quiet static but the signal is low. On the other hand walking has quiet visible pattern of signal that differs to going up or down or other activities.

Lag plots

At last we will use lag plots to confirm correlation of signal per activity and also to evaluate whether the values of our data are random. If the data are random, the lag plot will exhibit no identifiable pattern. If the data are not random, the lag plot will demonstrate a clearly identifiable pattern. And as we can see our data display linear pattern with autocorellation that follow the diagonal. We have divided the activities by their pattern of static or dynamic position for better understanding the data.



As we know now the data are correlated in both cases and displays pattern of signal. We can now move on and start to build our algorithm.

Algorithm building

Starting with data partition as our preparation for algorithm building. We divide the data to 60 % of train set and 40 % of test set. We do so, to reduce the amount of time required for computation. However even after this reduction, the computation can take a lot of time (On my computer with processor AMD Ryzen 3600 with 16G RAM took the decision tree 0,5h, kNN 3-4h and Rf 40 minutes.)

We will build three algorithms. Starting with decision tree, following k-nearest neighbors (kNN) and last will be random forest (Rf). For decision tree and kNN we use Caret package and for Rf we use ranger package because of it's ability to use most of the computational power. However to speed up the process with carate package we will use the doParralel package that allows to separate the computation into multiple cores.

Decision tree

We start with decision tree because it's fastest application and less demanding of computational power. A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules. However we wont display the decision tree itself because of necessary "tests" on attributes. It would take a lot of space to display all the branches.

We build our tree with complexity parameter (cp) tuning from 0.01 to 0.1. After tuning the parameter we fit the best one into our model. Also we define 10-fold cross validation function that we will use now and later for kNN.

```
control <- trainControl(method = "cv", number = 10, p = .9)

### Using doParralel package to use more cores in computer (This is set to 10 because of 12 maximum)
### It's gonna be used even for knn
cl <- makePSOCKcluster(10)
registerDoParallel(cl)

train_tree <- caret::train(activity ~ x + y + z + vm,
                           method = "rpart",
                           data = train_set,
                           tuneGrid = expand.grid(cp = seq(0.01, 0.1, 25)),
                           trControl = control)
```

Best performing value of complexity parameter is shown below. We fit it into our model:

```
train_tree
```

```
## CART
##
## 807731 samples
##      4 predictor
##      7 classes: 'working', 'stand_walk_down', 'standing', 'walking', 'going_up_down', 'walk+talk', '
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 726958, 726960, 726958, 726957, 726959, 726959, ...
## Resampling results:
##
##   Accuracy   Kappa
```

```
##   0.8803228  0.8401133
##
## Tuning parameter 'cp' was held constant at a value of 0.01
```

```
fit_tree <- caret::train(activity ~ x + y + z + vm,
                         method = "rpart",
                         data = train_set,
                         tuneGrid = data.frame(cp = train_tree$bestTune),
                         trControl = control)
```

The accuracy of regression tree is:

```
acc_tree <- confusionMatrix(caret::predict.train(fit_tree, test_set),
                            test_set$activity)$overall["Accuracy"]


acc_tree
```

```
##  Accuracy
## 0.8801744
```

That is quiet good but the performance can be deffinitely better with more complex model.

k-nearest neighbors

The kNN is a non-parametric method used for classification and regression. The algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. It uses 'feature similarity' to predict the values of new data points which further means that the new data point will be assigned a value based on how closely it matches the points in the training set. Since the algorithm is based on distance, normalizing data can improve performance. That's why we have done it on the begging.

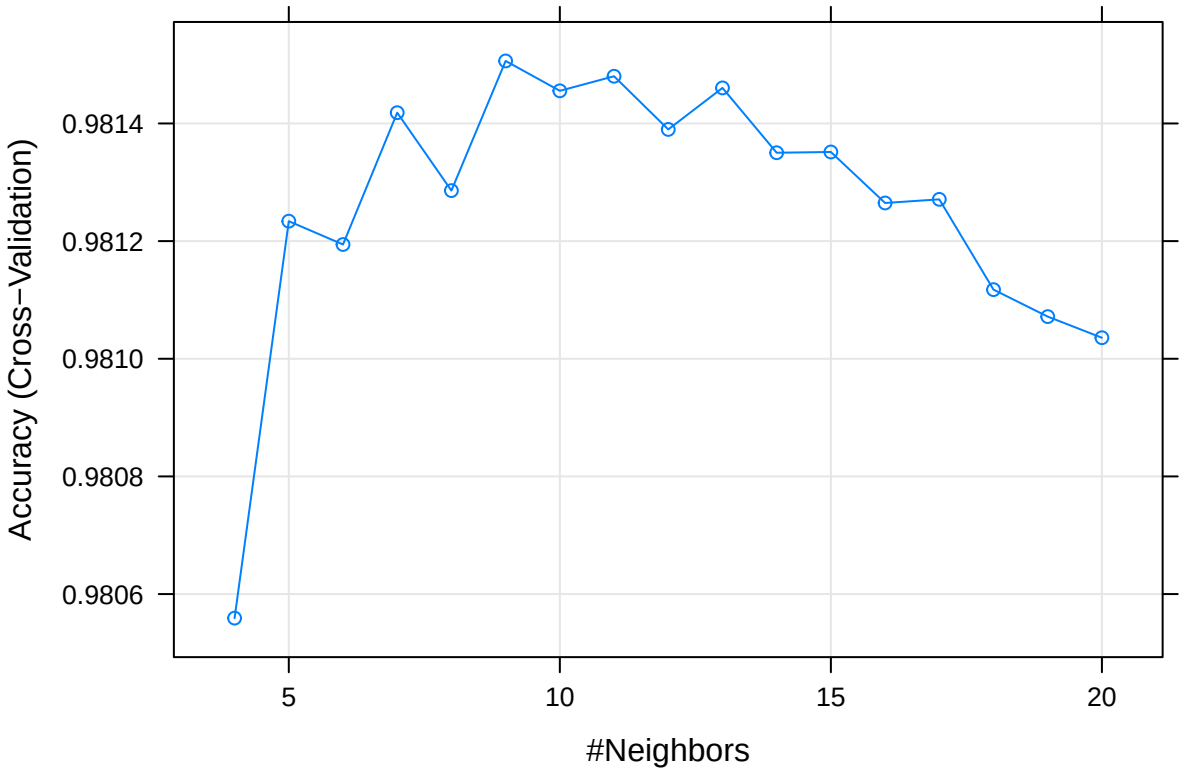
We tune the model with K sequence and with cross validation. After we find best K value we fit it into our model.

```
### Using doParralel from previous chunk
cl <- makePSOCKcluster(10)
registerDoParallel(cl)

train_knn <- caret::train(activity ~ x + y + z + vm, method = "knn",
                    data = train_set,
                    tuneGrid = data.frame(k = seq(4,20,1)),
                    trControl = control)
```

The plot of possible K values is displayed below. And we fit the optimal one into our model.

```
plot(train_knn, highlight = TRUE)
```



```
fit_knn <- caret::train(activity ~ x + y + z + vm, method = "knn",
                        data = train_set,
                        tuneGrid = data.frame(k = train_knn$bestTune),
                        trControl = control)
```

The accuracy of kNN with best tune k parameter is:

```
acc_knn <- confusionMatrix(caret::predict.train(fit_knn, test_set),
                           test_set$activity)$overall["Accuracy"]
acc_knn
```

```
##  Accuracy
## 0.9814667
```

It performs really well, but lets try our last algorithm.

Random forest

Random forest is a classifier that evolves from decision trees. It actually consists of many decision trees. To classify a new instance, each decision tree provides a classification for input data. It collects the classifications and chooses the most voted prediction as the result.

The Ranger package provides auto tuning function so we will use it. Then we fit the best parameters into our model:

```
## Preparing our data = clearing
data_rf <- train_set %>% select(-id, -seq) %>% as.data.frame()

## Creating classification task that is needed for tuneRanger
data.task <- makeClassifTask(data = data_rf, target = "activity")

### Auto tuning withe tuneRanger function
tune_rf <- tuneRanger(data.task, measure = list(acc), num.trees = 150,
                      tune.parameters = c("mtry", "min.node.size", "sample.fraction"))
```

We can see the best values below. So we fit them into the model.

```
tune_rf
```

```
## Recommended parameter settings:
##   mtry min.node.size sample.fraction
## 1    2             2       0.6657858
## Results:
##         acc exec.time
## 1 0.9820767    46.224
```

```
## Fitting in the best values
fit_rf <- ranger(activity ~ x + y + z + vm, train_set,
                 num.trees = 150,
                 mtry = 2,
                 min.node.size = 2,
                 sample.fraction = 0.6463493)
```

The accuracy of Rf is:

```
acc_rf <- confusionMatrix(as.factor(predict(fit_rf, test_set)$predictions),
                          test_set$activity)$overall["Accuracy"]

acc_rf
```

```
## Accuracy
## 0.982126
```

Results

The results of our algorithms are presented below:

method	Accuracy
Decision tree	0.8801744
k-nearest neighbors algorithm	0.9814667
Random Forest	0.9821260

As we can see the Random forest and kNN performance is really good. But the Random Forest perform little bit better. So we will predict the validation - unseen set, with our Random forrest model. You can see the resulting confusion matrix on the next page:

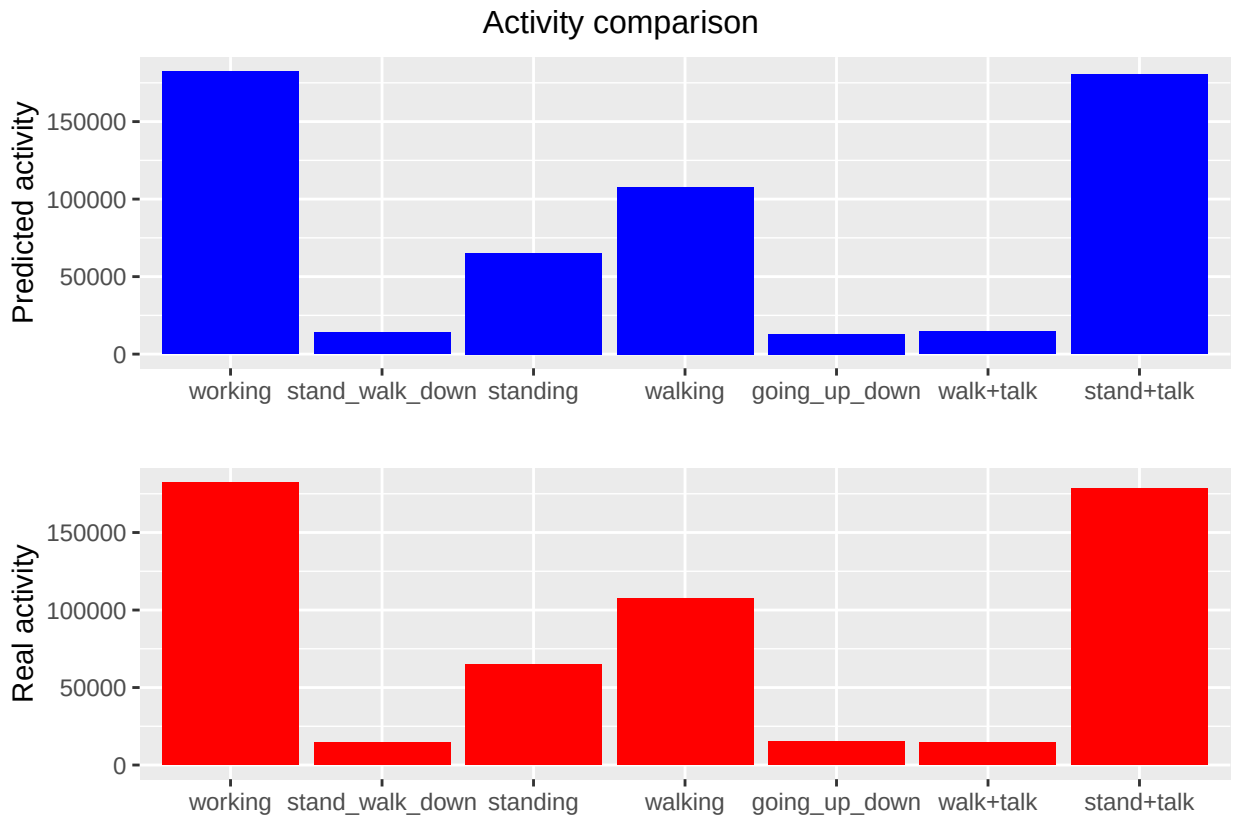
```r
predict <- predict(fit_rf, validation)

confusionMatrix(as.factor(predict$predictions), validation$activity)
```

```
## Confusion Matrix and Statistics
##
##                  Reference
## Prediction        working stand_walk_down standing walking going_up_down
##   working          182504               0        0       0             0
##   stand_walk_down       0           13768        7     162            25
##   standing              4               5    60817      47          3585
##   walking               3             506      132  106836            72
##   going_up_down         0              18     1608      20         11116
##   walk+talk             0              11       65      60            54
##   stand+talk            2              20     2030     101           611
##                  Reference
## Prediction        walk+talk stand+talk
##   working                 0          0
##   stand_walk_down         4          8
##   standing               51        682
##   walking               135         80
##   going_up_down          25        130
##   walk+talk           14231          6
##   stand+talk             19     177395
##
## Overall Statistics
##
##                Accuracy : 0.9822
##                  95% CI : (0.9818, 0.9825)
##     No Information Rate : 0.3163
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9764
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: working Class: stand_walk_down Class: standing
## Sensitivity                  1.0000                0.96092          0.9406
## Specificity                  1.0000                0.99963          0.9915
## Pos Pred Value               1.0000                0.98526          0.9329
## Neg Pred Value               1.0000                0.99901          0.9925
## Prevalence                   0.3163                0.02483          0.1121
## Detection Rate               0.3163                0.02386          0.1054
## Detection Prevalence         0.3163                0.02422          0.1130
## Balanced Accuracy            1.0000                0.98027          0.9660
##                      Class: walking Class: going_up_down Class: walk+talk
## Sensitivity                  0.9964              0.71888          0.98382
## Specificity                  0.9980              0.99679          0.99965
## Pos Pred Value               0.9914              0.86057          0.98641
## Neg Pred Value               0.9992              0.99229          0.99958
## Prevalence                   0.1858              0.02680          0.02507
```

```
## Detection Rate               0.1852              0.01927          0.02467
## Detection Prevalence         0.1868              0.02239          0.02501
## Balanced Accuracy            0.9972              0.85783          0.99174
##                       Class: stand+talk
## Sensitivity                      0.9949
## Specificity                      0.9930
## Pos Pred Value                   0.9846
## Neg Pred Value                   0.9977
## Prevalence                       0.3090
## Detection Rate                   0.3075
## Detection Prevalence             0.3123
## Balanced Accuracy                0.9940
```

As we can see on the confusion matrix, the model looks really great with accuracy over 98 %. We can also visualize it in histogram form:



However the 2% difference is really small so it's hard to see. But we can find out the balanced accuracy in confusion matrix above.

Conclusion

As we can see in the results the random forest performs great on our data with accuracy over 98 %. The hardest predicted activities are standing up, walking and going up or down stairs; standing; going up or down stairs. The first and last case is quiet understandable to have bad performance because of the changing movement pattern. However the standing activity should be predicted well, but in this case it could be misleading because we don't know if the standing pattern was like standing still or standing while moving the chest in different dimensions.

The model could be improved by setting the kNN and Rf together, however the ranger package creates its own class and I didn't found out how connect it. Expect this approach, other machine learning methods could be used for example deep learning or neural network.

The limitations of our work could be the device and sampling frequency that was used for recording. When we would fit our model to other devices it could work differently.

References

Casale, P. Pujol, O. and Radeva, P. 'Personalization and user verification in wearable systems using biometric walking patterns' Personal and Ubiquitous Computing, 16(5), 563-580, 2012

https://www.sciencedirect.com/topics/engineering/random-forest

https://nwfsc-timeseries.github.io/atsa-labs/sec-tslab-correlation-within-and-among-time-series.html